Introducing the GCC to the Polyhedron Model

Michael Claßen

University of Passau

St. Goar, June 30th 2009

Agenda

The GRAPHITE Project

- part of GCC 4.4
- goals:
  - provide interface to Polytope Model
  - cover as much “real world code” as possible
- in this talk: overview about status and code coverage experiments

Definitions

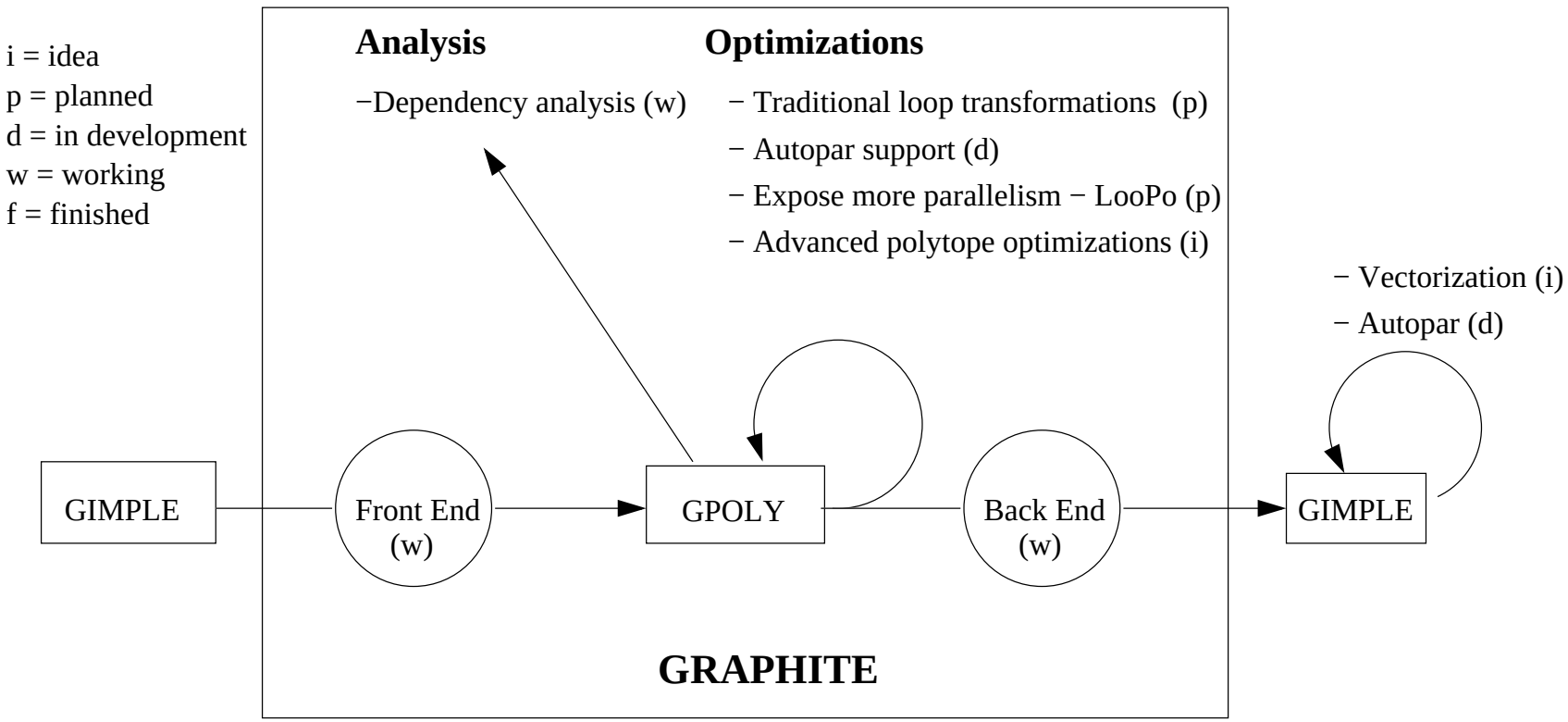
GIMPLE

- intermediate code representation in GCC
- common abstract language of all language frontends (e.g. C/C++, Fortran, Java)

GRAPHITE

- interface for polyhedra representation of GIMPLE
- goal: more high level loop optimizations

Status of GRAPHITE



What code can be represented?

- Structured code
- Affine loop bounds (e.g. `i < 4*n+3*j-1`)
- Constant loop strides (e.g. `i += 2`)
- Conditions contain comparisions ($<, <=, >, >=, ==, !=$) between affine functions
- Affine array accesses (e.g. `A[3i+1]`)

Analysis is working on GIMPLE, so the textual representation does not matter $\rightarrow$ hand made goto based loops work as well.

GPOLY

SCoP The optimization unit (e.g. a loop with some statements)
scop := ([*black box*])

black box An operation (e.g. statement) where only the memory accesses are known
black box :=
(*iteration domain*, *scattering matrix*, [*data reference*])

iteration domain The set of loop iterations for the black box

scattering matrix Defines the execution order of statement iterations (e.g. schedule)

data reference The memory cells accessed by the black box

Possible optimizations in GPOLY

- Iteration domain
  - Remove statement iterations
- Scattering matrix
  - Change the execution order of statement iterations to improve cache locality
  - ... expose parallelism (for vectorizer or autopar)
- Data reference (not yet supported)
  - Change the data layout to improve cache behaviour
  - ... to save memory.
  - Add additional memory to allow more parallelism

Coverage of GRAPHITE

Is it worth to write optimizations using GRAPHITE?

Code coverage

Compile time coverage:

- Count GIMPLE statements, loop headers and conditions
- Items are covered, if they are part of an "interesting" SCoP
- **Coverage**: ratio of covered items to total number of items

Hot spot coverage:

- One reference run $\longrightarrow$ count loop iterations
- Scale number of stmts, loops and conditions by number loop iterations
- $\Rightarrow$ Hot spots are taken into account $\Rightarrow$ more realistic metric

```
a = 10;
for (i=0; i<100; i++){
  b = 2 * i;
  if (b >= 0)
    c = 3;
}
```

Coverage:

	Compile	Hot Spot
loops	1/1	100/100
conds	2/2	200/200
stmts	7/9	700/702

```
bb0:
  a = 10;
  i = 0;

loop_head:
  b = 2 * i;
  if (b >= 0) goto then
  else goto exit_cond

then:
  c = 3;
  goto exit_cond

exit_cond:
  i++;
  if (i < 100) goto loop_head;
```

Code coverage GRAPHITE: SPEC 2006

Benchmark	Compile Time Coverage	Hotspot Coverage
bwaves	1.46	0.09
cactusADM	**3.54**	**99.41**
calculix	**5.28**	**76.09**
gromacs	2.77	2.20
h264ref	2.21	7.96
lbm	**35.79**	**0.02**
leslie3d	1.23	0.72
wrf	**7.70**	**54.86**
zeusmp	1.11	0.11
others (14)	< 1	< 1
average of all 23	**2.90**	**10.62**

Figure: Coverage May 2009 [in %]

Space for improvements?

Benchmark	Compile Time Coverage	Hotspot Coverage
base	2.90	**10.61**
SCEV_UNKNOWN	4.61	22.30
all conditions	18.50	19.58
all data structures	14.25	17.08
all structured control	48.43	**63.65**

Figure: Average coverage May 2009 [in %]

Conclusions

GRAPHITE

- Polytope Model part of GCC 4.4
- some simple optimizations implemented
- good code coverage

$\Longrightarrow$ ready to start implementing optimizations!

Thank You

wiki: `http://gcc.gnu.org/wiki/Graphite`

mailing list: `http://groups.google.com/group/gcc-graphite`